Relatório Projeto 2
Segurança Computacional 2023/2

Cifra de AES

Gabriel Nazareno Halabi - 150010290

INTRODUÇÃO

O objetivo deste projeto é a implementação de uma cifra que siga os padrões AES satisfatóriamente com os modos de operação ECB e CTR. As cifras da categoria Advanced Encryption Standard (AES) ou Rijndael, baseadas na cifra de blocos Rijndael utilizam o princípio da rede de substituição e permutação para implementar difusão nos dados criptografados, sendo aplicado em blocos de 128 bits e com chaves de 128 bits.

IMPLEMENTAÇÃO

Para a implementação da cifra, foi definida uma classe AESCypher na linguagem python que possui métodos referentes ás multiplas etapas da cifra. O primeiro passo é a realização de padding nos dados a serem cifrados, seguido da expansão da chave de 128 bits em uma coleção de 4 palavras de 32 bits para cada rodada que será realizada. Cada conjunto de palavras sofre primeiro uma rotação de elementos, seguida da substituição dos valores com base na tabela de substituição de bytes e por fim é realizada a operação XOR com a constante da rodada.

```
    def keyExpansion(self, key, rounds):
        temp = [ord(c) for c in key]
        words = [temp[i*4:i*4+4] for i in range(4)]

        rcon = [0, 1]
        for i in range(1, rounds):
            con = 2*rcon[i] if rcon[i] < 128 else 283^(2*rcon[i])
            rcon.append(con)
        for i in range(4,rounds*4+4):
            temp = words[i-1]
            if i%4 == 0:
                rot = temp[1:]+[temp[0]]
                sub = [self.sbox[0][b] for b in rot]
                temp = [sub[0]^rcon[int(i/4)]]+sub[1:]
            words.append([words[i-4][j]^temp[j] for j in range(4)])

        return words
```

Em seguida, cada bloco de 128 bits dos dados passa por 4 operações de transformação por rodada.

```
    def encryptBlock(self, block, keys, ctr):
        state = self.addRoundKey(block,keys[:4])

        for i in range(1,ctr+1):
            state = self.byteSubstitution(state)
            state = self.shiftRows(state)
            if i < ctr:
                state = self.mixColumns(state)
            state = self.addRoundKey(state, keys[i*4:i*4+4])

        return state
```

A primeira operação é a mais simples, realizando a operação XOR entre o bloco de dados e chave da rodada, gerada na expansão de chave realizada anteriormente.

```
    def addRoundKey(self, state, roundKey):
        return [[state[i][j]^roundKey[i][j] for j in range(4)] for i in range(4)]
```

Em seguida, é realizada a operação de substituição, utilizando a tabela de substituição de byte SBOX em cada um dos 16 bytes que compões o bloco.

```
    def byteSubstitution(self, state, reverse = False):
        state = [[self.sbox[reverse][b] for b in w] for w in state]

        return state
```

Os blocos são organizados como uma matriz 4x4 de bytes, e cada linha é deslocada por um fator diferente.

```
    def shiftRows(self, state, reverse = False):
        if reverse:
            temp = [[state[(4+j-i)%4][i] for i in range(4)] for j in range(4)]
        else:
            temp = [[state[(i+j)%4][i] for i in range(4)] for j in range(4)]

        return temp
```

Por fim, é realizada uma combinação dos valores de cada coluna entre si, com ajuda das tabelas de galois, resultando em valores dependentes dos resultados da rodada anterior

que auxiliam ainda mais na difusão.

```
    def mixColumns(self, state, reverse = False):
        temp = [[0]*4 for i in range(4)]
        if not reverse:
            galois = [[2,3,1,1],
                      [1,2,3,1],
                      [1,1,2,3],
                      [3,1,1,2]]
        else:
            galois = [[14,11,13,9],
                      [9,14,11,13],
                      [13,9,14,11],
                      [11,13,9,14]]

        for i in range(4):
            for j in range(4):
                for k in range(4):
                    n = self.moduloIrreductible(state[i][k], galois[j][k])
                    temp[i][j] ^= n

        return temp
```

Esta sequência de 4 operações é realizada para cada rodada requerida até completar a cifração. No caso do modo de operação CTR, a principal diferença é a aplicação de um nonce para uma série de blocos a serem cifrados com a chave em vez dos dados originais. Depois de cifrar este stream gerado é que é realizada uma operação XOR entre a stream cifrada os dados.

```
        if opmode == 'ECB':
            stream = data if dec else self.ISOPadding(data)
        elif opmode == 'CTR':
            stream = []
            for i in range(ceil(len(data)/16)):
                temp = nonce.copy()
                temp.extend([b for b in (i).to_bytes(8,'big')])
                stream.extend(temp)
        else:
            return
```

RESULTADOS

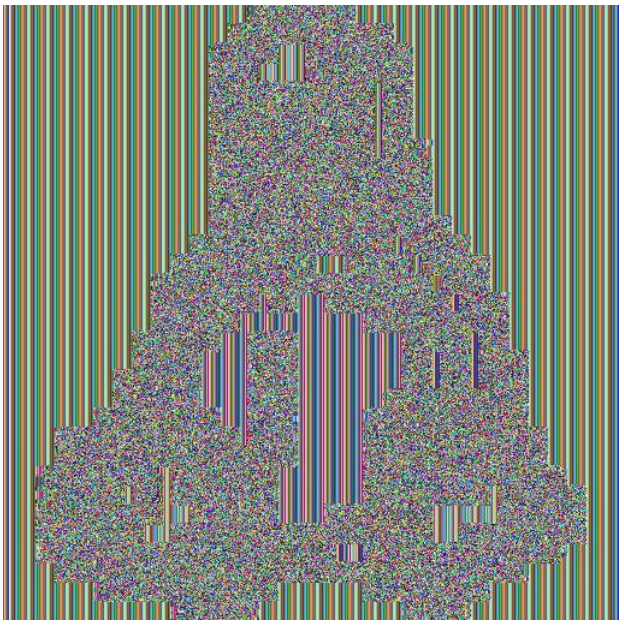

Fig 1. Imagem codificada em modo de operação ECB 10 Rodadas

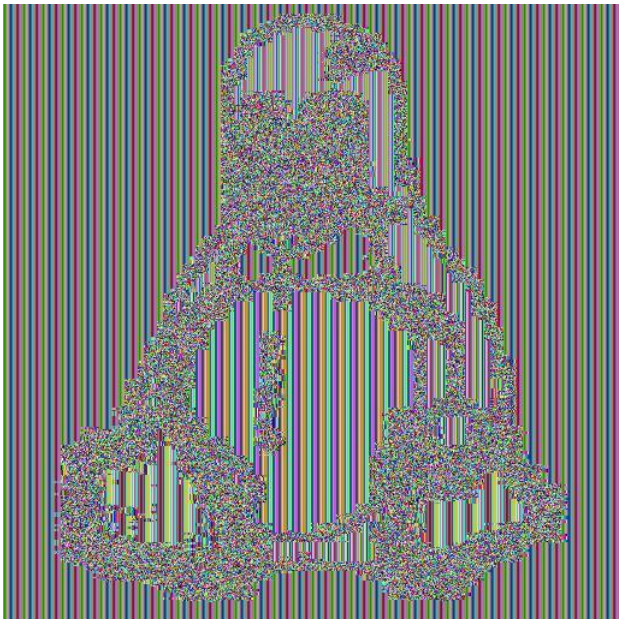

Fig 2. Imagem codificada em modo de operação CTR 1 Rodada

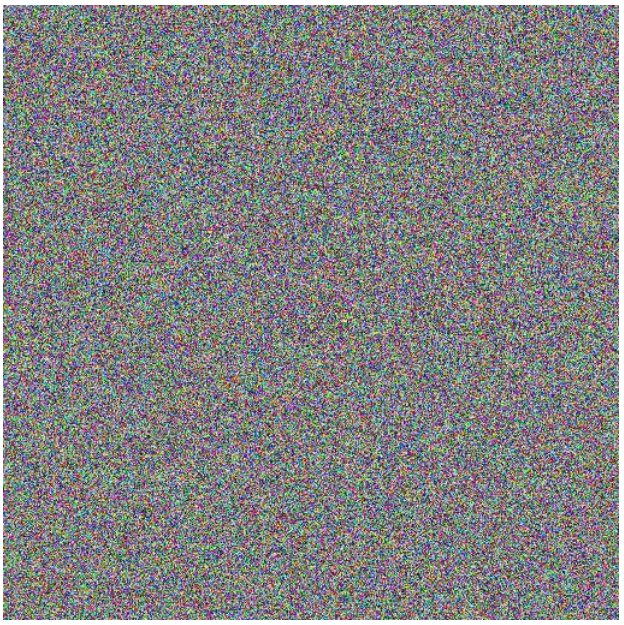

Fig 3. Imagem codificada em modo de operação CTR 5 Rodadas

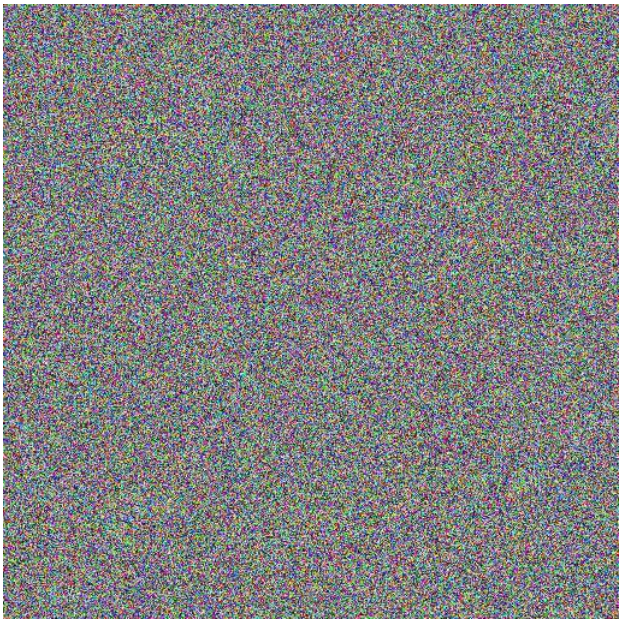

Fig 4. Imagem codificada em modo de operação CTR 9 Rodadas

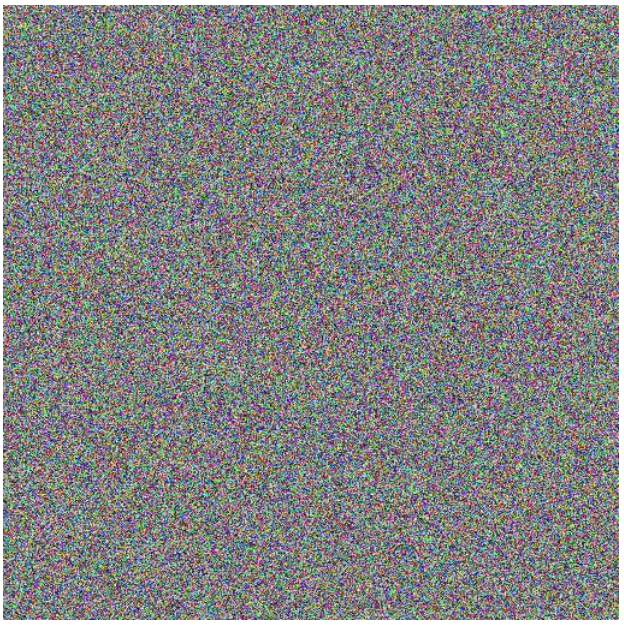

Fig 5. Imagem codificada em modo de operação CTR 13 Rodadas

CONCLUSÃO

O mode de operação CTR funciona de forma muito mais efetiva na difusão dos dados cifrados, como observado com os resultados obtidos, enquanto 10 rodadas no modo ECB resultaram em uma imagem que ainda retém informações sobre a imagem original, comparável a apenas uma rodada de CTR, quando aumentamos para 5 ou mais rodadas fica claro que as características da imagem original se tornam cada vez mais indistinguíveis.